Int. J. Appl. Math. Comput. Sci., 2017, Vol. 27, No. 4, 827–837
DOI: 10.1515/amcs-2017-0058

amcs

PROCEDURAL GENERATION OF AESTHETIC PATTERNS FROM DYNAMICS AND ITERATION PROCESSES

KRZYSZTOF GDAWIEC [a]

[a] Institute of Computer Science
University of Silesia, Będzińska 39, 41-200 Sosnowiec, Poland
e-mail: kgdawiec@ux2.math.us.edu.pl

Aesthetic patterns are widely used nowadays, e.g., in jewellery design, carpet design, as textures and patterns on wallpapers, etc. Most of the work during the design stage is carried out by a designer manually. Therefore, it is highly useful to develop methods for aesthetic pattern generation. In this paper, we present methods for generating aesthetic patterns using the dynamics of a discrete dynamical system. The presented methods are based on the use of various iteration processes from fixed point theory (Mann, S, Noor, etc.) and the application of an affine combination of these iterations. Moreover, we propose new convergence tests that enrich the obtained patterns. The proposed methods generate patterns in a procedural way and can be easily implemented on the GPU. The presented examples show that using the proposed methods we are able to obtain a variety of interesting patterns. Moreover, the numerical examples show that the use of the GPU implementation with shaders allows the generation of patterns in real time and the speed-up (compared with a CPU implementation) ranges from about 1000 to 2500 times.

Keywords: dynamical system, dynamics, iteration process, aesthetic pattern.

1. Introduction

Ornaments and regular tiling patterns have a long tradition in human culture. Their origins date back to ancient cultures like China, Egypt, Greece and Islam (von Gagern and Richter-Gebert, 2009). During these times people have replicated repeating patterns on floors, walls, ceilings and buildings, and in ceramics, fabrics, rugs, wallpapers and stained-glass windows. Today, we continue to be fascinated by symmetry and repetition in design. We ask whether the beauty of the symmetrical ornaments and the designs of various cultures can be simulated with the aid of a computer. One way is through the use of various mathematical equations (Pickover, 2001).

The literature is full of very diverse methods of aesthetic pattern generation that use mathematical equations in the generation process. One of the approaches employed in these methods is the dynamics of discrete dynamical systems. In this way, we are able to obtain patterns that possess wallpaper symmetries (Chung and Chan, 1993; Lu *et al.*, 2007), spherical symmetries (Chung and Chan, 1995), frieze symmetries (Lu *et al.*, 2010), Archimedean tilings (Ouyang *et al.*, 2015), etc. In recent years, another interesting approach has been proposed, namely, the use of iteration processes from fixed point theory. The processes were successfully applied in the generation of generalised Mandelbrot and Julia sets (Ashish *et al.*, 2014; Kang *et al.*, 2015b), in inversion fractals (Gdawiec, 2017) or in polynomiography (patterns obtained from the polynomial root-finding process) (Gdawiec *et al.*, 2015; Kang *et al.*, 2015a).

Besides methods that use various mathematical equations, we can find in the literature many other approaches to aesthetic pattern generation. The most popular one is the use of different types of grammars. Shape grammars are the most popular types among these methods. For instance, they were employed to generate ethnic Zhuang embroidery designs (Jia and Ming-Xi, 2013) and Islamic geometric patterns (Sayed *et al.*, 2016). Other types of grammars used to generate patterns are collage grammars (Klempien-Hinrichs and von Totth, 2010) and l-systems (Chen *et al.*, 2012). Additional popular approaches employed in the field include graph methods (Yeh *et al.*, 2013), cellular automata (Greenfield, 2016), neural networks (Setti, 2015) and Petri nets (Lalitha and Rangarajan, 2012). We can also find methods that are based on examples (Qi and Li, 2009; Wei *et al.*,

2009) and in which a user-driven planning strategy is applied (Anderson and Wood, 2008).

In this paper, we show how to use iteration processes from fixed point theory to alter the dynamics of methods that are based on the application of the dynamics of discrete dynamical systems in the generation of aesthetic patterns. The alternation of dynamics will lead to the change in a pattern's shape. Moreover, we show how to combine different iteration processes into one to obtain completely new patterns. In order to enrich further the obtained patterns, we also propose new convergence tests based on metric and non-metric conditions.

The remainder of this paper is outlined as follows. In Section 2, some basic information on generating symmetrical patterns from dynamics is presented. Next, in Section 3, modifications of the algorithm from Section 2 are presented. The first modification is based on the use of different iteration processes from fixed point theory together with their generalisation into vector parameters. The second modification uses an affine combination of iterations and the last one relies on the use of different convergence tests. Moreover, we present some remarks on how to implement the proposed algorithms on the GPU using shaders. Then, in Section 4, we present various examples of patterns obtained with the proposed methods. Also, we make a comparison of the generation times between the GPU and CPU implementations. Finally, Section 5 rounds off the discussion with some concluding remarks.

2. Symmetrical patterns from dynamics

Dynamical systems are mathematical models which contain the rules describing the way some quantity undergoes a change through time. Graphical presentations of the phase portraits not only reveal the complex behaviours of such systems, but also quite often have high artistic appeal (Chung *et al*., 2001). Many different methods of creating phase portraits with aesthetic appeal have been proposed (Chung and Chan, 1993; Chung *et al*., 2001; Lu *et al*., 2007; Ouyang *et al*., 2015). One of these (which will be used in this paper) has been proposed by Chung and Chan (1993).

In their work, the authors studied the following discrete dynamical system:

$$\begin{cases} x_{n+1} = x_n - f(x_n, y_n), \\ y_{n+1} = y_n - g(x_n, y_n), \end{cases} \tag{1}$$

where $(x_0, y_0)^T \in \mathbb{R}^2$ is a starting point (the transposition in the notation of the point is used because in the paper we assume the column notation of points, vectors, etc.) and $f, g : \mathbb{R}^2 \to \mathbb{R}$ are functions. They studied the conditions under which the phase portrait of this dynamical system will have plane symmetries (translation, reflection, glide reflective and rotational symmetry). For instance, they showed that if we want to obtain a pattern with a rotational symmetry with angle θ, then the functions f, g should fulfil the following conditions:

$$f(x'', y'') - 2\cos\theta f(x', y') + f(x, y) = 0, \tag{2}$$

where

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = R_\theta \begin{pmatrix} x \\ y \end{pmatrix}, \tag{3}$$

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = R_\theta \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{4}$$

and

$$g(x, y) = \frac{\cos\theta f(x, y) - f(x', y')}{\sin\theta}. \tag{5}$$

Moreover, using the conditions for plane symmetries, they derived conditions that f, g must meet in order to obtain patterns with wallpaper symmetries. For instance, the conditions to obtain a pattern with a p4 symmetry (a square lattice with a 4-fold symmetry, i.e., rotational symmetry with respect to a 90° angle; more information on the different types of wallpaper symmetries can be found, for example, in the work of Horne (2000)) are as follows:

$$f(x, y) = -f(-x, -y), \tag{6}$$

$$f(x, y) = f(x + P, y) = f(x, y + P), \tag{7}$$

$$g(x, y) = -f(-y, x), \tag{8}$$

where P is a period of the function.

In order to generate the patterns, Chung and Chan (1993) used the method presented in Algorithm 1. In the algorithm, each starting point $(x_0, y_0)^T$ in the area A is iterated using (1). The iteration process is performed as long as the number of iterations is smaller than a fixed number of iterations K or a convergence condition is not satisfied. When the iteration process ends, we assign a colour to the starting point based on the number of performed iterations and a chosen colourmap.

Figure 1 presents some examples of patterns obtained using Algorithm 1. The parameters used to generate these were the following:

(a) $A = [-15, 15]^2$, $K = 50$, $\varepsilon = 0.1$ and

$$\begin{cases} f(x, y) = \sin x \cos y, \\ g(x, y) = \cos 2x \sin y. \end{cases} \tag{9}$$

(b) $A = [-20, 20]^2$, $K = 50$, $\varepsilon = 0.1$ and

$$\begin{cases} f(x,y) = 2r\sin x\cos Cy + 2s\sin(h_1(x,-y)) \\ \qquad \cos(Ch_2(x,y)) + 2(s-r) \\ \qquad \sin(h_1(-x,-y))\cos(Ch_2(x,-y)), \\ g(x,y) = Bf(x,y) - C[2r\sin(h_1(x,-y)) \\ \qquad \cos(Ch_2(x,y)) + 2s\sin(h_1(-x,-y)) \\ \qquad \cos(Ch_2(x,-y)) - 2(s-r) \\ \qquad \sin x\cos Cy], \end{cases} \tag{10}$$

where

$$h_1(x,y) = 0.5x + Dy, \tag{11}$$

$$h_2(x,y) = Dx + 0.5y \tag{12}$$

and $B = \frac{1}{\sqrt{3}}$, $C = \frac{2}{\sqrt{3}}$, $D = \frac{\sqrt{3}}{2}$, $r = 0.2$, $s = 0.1$.

(c) $A = [-10, 10]^2$, $K = 50$, $\varepsilon = 0.1$ and

$$\begin{cases} f(x,y) = 0.2(\sin x\cos 2y - \sin(-x)\cos(-2y)) \\ \qquad + 0.1(\sin(-y)\cos 2x - \sin y\cos(-2x)), \\ g(x,y) = -0.2(\sin(-y)\cos 2x - \sin y\cos(-2x)) \\ \qquad - 0.1(\sin(-x)\cos(-2y) - \sin x\cos 2y). \end{cases} \tag{13}$$

(d) $A = [-15, 15]^2$, $K = 50$, $\varepsilon = 0.1$ and

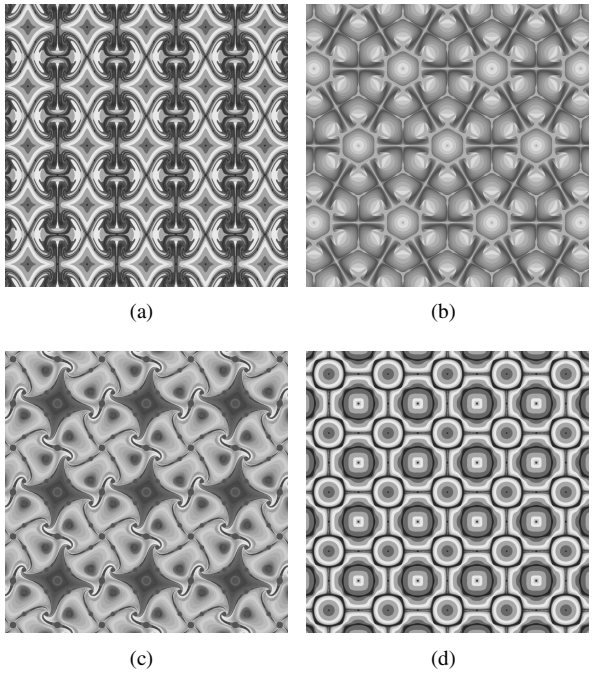
$$\begin{cases} f(x,y) = 0.5\sin 2x\cos y + 0.1\sin x\cos 2y, \\ g(x,y) = 0.5\cos x\sin 2y + 0.1\cos 2x\sin y. \end{cases} \tag{14}$$

Algorithm 1. Generating pattern from dynamics.

Require: $f, g : \mathbb{R}^2 \to \mathbb{R}$—functions, $A \subset \mathbb{R}^2$—area, K—maximum number of iterations, $\varepsilon > 0$—accuracy, $colourmap[0 \dots C-1]$—colourmap with C colours.

Ensure: pattern in the area A.

for $(x_0, y_0)^T \in A$ **do**
  $n = 0$
  while $n < K$ **do**
    $x_{n+1} = x_n - f(x_n, y_n)$
    $y_{n+1} = y_n - g(x_n, y_n)$
    if $\sqrt{(x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2} < \varepsilon$ **then**
      break
    end if
    $n = n + 1$
  end while
  $i = \lfloor (C-1)\frac{n}{K} \rfloor$
  colour (x_0, y_0) with $colourmap[i]$
end for



(a) (b) (c) (d)

Fig. 1. Examples of patterns obtained using the dynamics of (1).

3. Iteration processes and dynamics

Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be defined as follows:

$$T(p) = p - \begin{pmatrix} f(p) \\ g(p) \end{pmatrix} = \begin{pmatrix} x_p - f(p) \\ y_p - g(p) \end{pmatrix}, \tag{15}$$

where $p = (x_p, y_p)^T \in \mathbb{R}^2$ and $f, g : \mathbb{R}^2 \to \mathbb{R}$ are functions. Then, the dynamical system (1) can be written as

$$p_{n+1} = T(p_n). \tag{16}$$

This form of feedback iteration is also called the Picard iteration and it is used in many different algorithms, e.g., numerical polynomial root finding (Gdawiec *et al.*, 2015).

One of the most important applications of Picard's iteration is finding the fixed points of a contractive mapping using the Banach fixed point theorem. Fixed point theory includes many other methods that allow one to find fixed points of a given mapping. Most of them rely on the feedback iteration of the form, other than Picard's iteration.

Let us recall some of them. Let (X, d) be a metric space, $T : X \to X$ be a mapping and $p_0 \in X$ be a starting point.

1. The Mann iteration (Mann, 1953), which is a one-step iteration defined as follows:

$$p_{n+1} = (1 - \alpha_n)p_n + \alpha_n T(p_n), \tag{17}$$

where $\alpha_n \in (0, 1]$ for all $n \in \mathbb{N}$.

2. The S-iteration (Agarwal *et al.*, 2007), which is a two-step iteration defined as follows:

$$\begin{cases} p_{n+1} = (1-\alpha_n)T(p_n) + \alpha_n T(u_n), \\ u_n = (1-\beta_n)p_n + \beta_n T(p_n), \end{cases} \tag{18}$$

where $\alpha_n \in (0,1]$, $\beta_n \in [0,1]$ for all $n \in \mathbb{N}$.

3. The Noor iteration (Noor, 2000), which is a three-step iteration defined as follows:

$$\begin{cases} p_{n+1} = (1-\alpha_n)p_n + \alpha_n T(u_n), \\ u_n = (1-\beta_n)p_n + \beta_n T(v_n), \\ v_n = (1-\gamma_n)p_n + \gamma_n T(p_n), \end{cases} \tag{19}$$

where $\alpha_n \in (0,1]$, $\beta_n, \gamma_n \in [0,1]$ for all $n \in \mathbb{N}$.

When we look at the different iterations, we see that they can have different numbers of parameters (one, two, three or even more) and varying complexity, from very simple as in the Mann iteration to more complex as in the S-iteration. We can also observe that the iterations combine, in various ways, the point from the previous iteration and the transformed points computed in the current one (compare the Noor and S iterations). Moreover, we notice that the iterations for particular parameter values can reduce to other iterations. For instance, if we take in the Noor iteration $\gamma_n = 0$, $\beta_n = 0$ for all $n \in \mathbb{N}$, we obtain a Mann iteration. A review of various iterations and their dependencies can be found in the work of Gdawiec and Kotarski (2017).

As noted at the beginning of this section, the method for generating symmetrical patterns from Section 2 uses the Picard iteration. This type of iteration is used to get the dynamics of a given discrete dynamical system. In order to obtain new patterns from the same functions (used to define the dynamical system), we propose to alter the dynamics of this dynamical system by using different iterations from fixed point theory other than Picard's iteration.

In the iterations, the parameters are real numbers that belong to $(0,1]$ or $[0,1]$. This is one of the assumptions used in fixed point theory to prove the convergence (weak, strong) of iterations to a fixed-point in different spaces (metric, Banach, Hilbert, etc.). As we are interested in generating new patterns by changing the dynamics, rather than converging to a fixed point, we can omit these assumptions and take parameters outside of those intervals. Moreover, we can extend the parameters even further, to two-dimensional vectors.

Let $\mathbf{1} = (1,0)^T$ and $\alpha_n = (x_{\alpha_n}, y_{\alpha_n})^T$, $\beta_n = (x_{\beta_n}, y_{\beta_n})^T$, $\gamma_n = (x_{\gamma_n}, y_{\gamma_n})^T$, etc. In the iterations, we need subtraction and multiplication of the parameters. We define these arithmetic operations in a similar way as in the case of complex numbers, i.e.,

$$(x_1,y_1)^T \pm (x_2,y_2)^T = (x_1 \pm x_2, y_1 \pm y_2)^T, \tag{20}$$

$$\begin{aligned}(x_1,y_1)^T \cdot (x_2,y_2)^T \\ = (x_1x_2 - y_1y_2, x_1y_2 + y_1x_2)^T,\end{aligned} \tag{21}$$

where $(x_1,y_1)^T, (x_2,y_2)^T \in \mathbb{R}^2$. Having established the basic operations, we are able to extend the iterations to vector parameters. For instance, the Noor iteration takes the following form:

$$\begin{cases} p_{n+1} = (\mathbf{1}-\alpha_n)\cdot p_n + \alpha_n \cdot T(u_n), \\ u_n = (\mathbf{1}-\beta_n)\cdot p_n + \beta_n \cdot T(v_n), \\ v_n = (\mathbf{1}-\gamma_n)\cdot p_n + \gamma_n \cdot T(p_n), \end{cases} \tag{22}$$

where $\alpha_n, \beta_n, \gamma_n \in \mathbb{R}^2$ for all $n \in \mathbb{N}$.

As mentioned in Section 2, Chung and Chan (1993) studied the conditions under which the phase portrait of (1) has plane and wallpaper symmetries. Analogous analysis can be made for (15) and the iteration processes. Each of the iteration processes must be studied independently because, as they have different numbers of parameters and diverse complexity, the conditions will be different.

Let us consider two iteration processes—the Mann and the S-iteration. We will derive conditions that functions f and g must fulfil to get the translation symmetry, assuming that the phase portrait has a period T along the x-axis, i.e., translation symmetry. The phase portrait is invariant after the transformation $p' = (x_p - T, y_p)^T$, where $p = (x_p, y_p)^T$.

Using (15), the Mann iteration takes the following form:

$$\begin{aligned} p_{n+1} &= (\mathbf{1}-\alpha_n)p_n + \alpha_n T(p_n) \\ &= p_n - \alpha_n p_n + \alpha_n p_n - \alpha_n \begin{pmatrix} f(p_n) \\ g(p_n) \end{pmatrix} \\ &= p_n - \alpha_n \begin{pmatrix} f(p_n) \\ g(p_n) \end{pmatrix}. \end{aligned} \tag{23}$$

Thus, using the Mann iteration, we obtain a very similar formula to the one found with the Picard iteration. The only difference is that in the Mann iteration we have a scaling factor α_n. This factor does not affect the symmetry conditions derived by Chung and Chan (1993). Thus, the conditions under which to obtain the translation symmetry considered are the following:

$$f(x,y) = f(x+T,y), \tag{24}$$

$$g(x,y) = g(x+T,y). \tag{25}$$

Now, let us consider the case with the S-iteration. The formula for u_n is the same as in the case of the Mann iteration:

$$u_n = p_n - \beta_n \begin{pmatrix} f(p_n) \\ g(p_n) \end{pmatrix}, \tag{26}$$

whereas the formula for p_{n+1} takes the following form:

$$\begin{aligned} p_{n+1} &= (\mathbf{1}-\alpha_n)T(p_n)+\alpha_n T(u_n) \\ &= (\mathbf{1}-\alpha_n)\left[p_n-\begin{pmatrix} f(p_n) \\ g(p_n)\end{pmatrix}\right] \\ &\quad +\alpha_n\left[p_n-\beta_n\begin{pmatrix} f(p_n) \\ g(p_n)\end{pmatrix}-\begin{pmatrix} f(u_n) \\ g(u_n)\end{pmatrix}\right] \\ &= p_n-\left[(\mathbf{1}-\alpha_n+\alpha_n\beta_n)\begin{pmatrix} f(p_n) \\ g(p_n)\end{pmatrix}\right. \\ &\quad \left.+\alpha_n\begin{pmatrix} f(u_n) \\ g(u_n)\end{pmatrix}\right]. \end{aligned} \tag{27}$$

Substituting $(x_{p'}, y_{p'}) = (x_p - T, y_p)^T$ into (26) and (27), we have

$$u'_n = \begin{pmatrix} x_{p'_n}+T \\ y_{p'_n}\end{pmatrix}-\beta_n\begin{pmatrix} f(x_{p'_n}+T, y_{p'_n}) \\ g(x_{p'_n}+T, y_{p'_n})\end{pmatrix} \tag{28}$$

and

$$\begin{aligned} p'_{n+1} = \begin{pmatrix} x_{p'_n} \\ y_{p'_n}\end{pmatrix} &- \left[(\mathbf{1}-\alpha_n+\alpha_n\beta_n)\right. \\ &\left.\begin{pmatrix} f(x_{p'_n}+T, y_{p'_n}) \\ g(x_{p'_n}+T, y_{p'_n})\end{pmatrix}+\alpha_n\begin{pmatrix} f(u'_n) \\ g(u'_n)\end{pmatrix}\right]. \end{aligned} \tag{29}$$

For (27) and (29) to be identical for all n, we must have

$$\begin{aligned} (\mathbf{1}-\alpha_n+\alpha_n\beta_n)&\begin{pmatrix} f(x_p,y_p)-f(x_p+T,y_p) \\ g(x_p,y_p)-g(x_p+T,y_p)\end{pmatrix} \\ &+\alpha_n\begin{pmatrix} f(u_n)-f(u'_n) \\ g(u_n)-g(u'_n)\end{pmatrix}=0, \end{aligned} \tag{30}$$

where

$$u_n = \begin{pmatrix} x_p \\ y_p\end{pmatrix}-\beta_n\begin{pmatrix} f(x_p,y_p) \\ g(x_p,y_p)\end{pmatrix}, \tag{31}$$

$$u'_n = \begin{pmatrix} x_p+T \\ y_p\end{pmatrix}-\beta_n\begin{pmatrix} f(x_p+T,y_p) \\ g(x_p+T,y_p)\end{pmatrix}. \tag{32}$$

As we can see, the conditions for the translation symmetry using the S-iteration become more complex than in the case of the Picard and Mann iterations. After studying the conditions for other types of symmetry, we can make a similar observation. Thus, the complexity of the iteration method has an impact on that of the symmetry conditions—the more complex the iteration, the more complex the conditions.

Using iterations from fixed point theory together with the mapping (15), we are able to create new patterns, but we have limited possibilities of obtaining these because we use only one mapping and iteration. Thus, it would be advantageous to be able to generate new patterns from several mappings and iterations. To this end, we can use an affine combination. Let $T_1, T_2, \ldots, T_N$ be mappings given by (15) and $I_{v_1}, I_{v_2}, \ldots, I_{v_N}$ be iterations (Picard, Mann, Noor, etc.), where $N \geq 1$ and v_i for $i = 1, 2, \ldots, N$ is a vector of parameters of the iteration method. Moreover, for each iteration we fix $\lambda_i \in \mathbb{R}$ for $i = 1, 2, \ldots, N$ such that $\sum_{i=1}^{N} \lambda_i = 1$. Now, we define a combined iteration process as follows:

$$p_{n+1} = \sum_{i=1}^{N} \lambda_i I_{v_i}(p_n, T_i). \tag{33}$$

If we take $N = 1$ and $\lambda_1 = 1$, then (33) reduces to the case with a single iteration and mapping. This is a general form of the iteration process that will be used in the paper. In the sequel, if we refer to a single iteration, then we make the implicit assumption that $N = 1$ and $\lambda_1 = 1$.

In Algorithm 1, we can modify not only the iteration process, but also the convergence condition (test). In the condition from Algorithm 1, we check if the Euclidean distance between two consecutive points is smaller than the given accuracy ε. This is a standard convergence test used in many numerical algorithms, e.g., numerical solution of non-linear equations. Gdawiec (2013) presented new convergence tests using the polynomial root-finding methods in order to obtain new artistic patterns. Against this background, we also constructed various convergence tests for the generation algorithm that uses dynamical systems.

1. The Pickover test (Pickover, 2001):

$$|x^2_{p_{n+1}} + y^2_{p_{n+1}} - (x^2_{p_n} + y^2_{p_n})| < \varepsilon, \tag{34}$$

where $p_{n+1} = (x_{p_{n+1}}, y_{p_{n+1}})^T$, $p_n = (x_{p_n}, y_{p_n})^T$.

2. The fractional distance test:

$$\left(|x_{p_{n+1}} - x_{p_n}|^q + |y_{p_{n+1}} - y_{p_n}|^q\right)^{\frac{1}{q}} < \varepsilon, \tag{35}$$

where $p_{n+1} = (x_{p_{n+1}}, y_{p_{n+1}})^T$, $p_n = (x_{p_n}, y_{p_n})^T$ and $q \in (0, 1]$.

3. The maximum distance with weights test:

$$\max\{a|x_{p_{n+1}} - x_{p_n}|, b|y_{p_{n+1}} - y_{p_n}|\} < \varepsilon, \tag{36}$$

where $p_{n+1} = (x_{p_{n+1}}, y_{p_{n+1}})^T$, $p_n = (x_{p_n}, y_{p_n})^T$ and $a, b \in \mathbb{R}_+$ are the weights.

4. The alternative test (Gdawiec, 2013):

$$|a(x_{p_{n+1}} - x_{p_n})| < \varepsilon_1 \vee |b(y_{p_{n+1}} - y_{p_n})| < \varepsilon_2, \tag{37}$$

where $p_{n+1} = (x_{p_{n+1}}, y_{p_{n+1}})^T$, $p_n = (x_{p_n}, y_{p_n})^T$ and $a, b \in \mathbb{R}$.

5. The difference test:

$$|(a|x_{p_{n+1}} - x_{p_n}|)^q - (b|y_{p_{n+1}} - y_{p_n}|)^r| < \varepsilon, \quad (38)$$

where $p_{n+1} = (x_{p_{n+1}}, y_{p_{n+1}})^T$, $p_n = (x_{p_n}, y_{p_n})^T$, $a, b \in \mathbb{R}$ and $q, r \in \mathbb{R}_+$.

All the presented modifications of Algorithm 1 are summarized in Algorithm 2. We can implement this algorithm on the GPU. For this purpose, we can use either the OpenCL, CUDA libraries or the shaders employed by graphics libraries like OpenGL or Direct3D. The OpenCL and CUDA libraries are normally used in general-purpose computations, whereas shaders were introduced for graphical applications, such as the rendering of 3D graphics. As the proposed algorithm is purely graphical and the patterns can be used in a 3D scene as textures generated in a procedural way, the use of shaders is a more obvious choice. Of course, the OpenCL or CUDA libraries may be also used.

Let us turn now to the use of OpenGL and OpenGL Shading Language (GLSL) in the implementation of Algorithm 2 on the GPU. Each point in the area A considered is calculated independently from the others, which makes the algorithm easy to parallelize using shaders. All computations are made in the fragment shader. Some of the input parameters of the algorithm can be sent to the fragment shader using uniform variables. For instance, we can send the following parameters in this way: the area, the maximum number of iterations, a colour map and an affine combination coefficients. Parameters like functions defining mappings T_i, iterations or convergence tests cannot be sent as simple uniform variables precisely because they are functions. For this purpose, we can use GLSL subroutines. A known issue in the generation of patterns in a procedural way is aliasing (Ebert *et al.*, 2002). In order to deal with this problem, we can use the super sampling method, which is widely employed in many programs for generating fractal art patterns, e.g., Fractint, ChaosPro, Ultra Fractal.

Algorithm 2. Generating pattern from dynamics using a combination of iterations.

Require: $f_1, f_2, \ldots, f_N, g_1, g_2, \ldots, g_N : \mathbb{R}^2 \to \mathbb{R}$—functions defining mappings $T_1, T_2, \ldots, T_N$ of the form (15), $A \subset \mathbb{R}^2$—area, K—maximum number of iterations, $I_{v_1}, I_{v_2}, \ldots, I_{v_N}$—iterations, $\lambda_1, \lambda_2, \ldots, \lambda_N$—affine combination coefficients, $C_u : \mathbb{R}^2 \times \mathbb{R}^2 \to \{true, false\}$—convergence test, $colourmap[0 \ldots C-1]$—colour map with C colours.

Ensure: pattern in the area A.

```
for p_0 ∈ A do
    n = 0
    while n < K do
        p_{n+1} = Σ_{i=1}^{N} λ_i I_{v_i}(p_n, T_i)
        if C_u(p_n, p_{n+1}) = true then
            break
        end if
        n = n + 1
    end while
    i = ⌊(C − 1) n/K⌋
    colour p_0 with colourmap[i]
end for
```

4. Examples

In this section, we present some examples of aesthetic patterns obtained with the methods proposed in this paper. In all examples, we use supersampling anti-aliasing with factor 4.

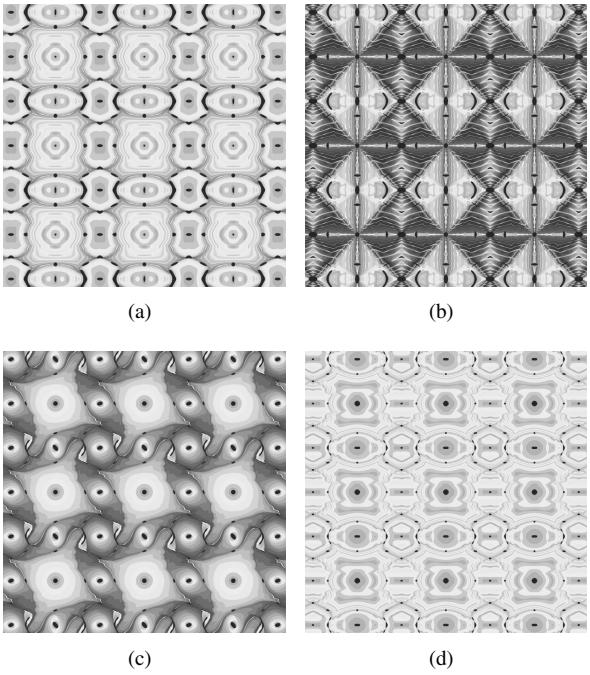
For comparison purposes, the proposed methods were implemented in the GLSL (GPU implementation) and in a Java-based programming language named *Processing* (CPU implementation). All patterns were rendered to images with a resolution of 800×800 pixels. Thus, taking into account the anti-aliasing process (factor equal to 4) and the final resolution (800×800 pixels) the images were initially rendered using a resolution of 3200×3200 pixels. All the experiments were performed on a computer with the following characteristics: an Intel i5-4570 (@ 3.2 GHz) processor, 16 GB DDR3 RAM, AMD Radeon HD7750 with 1 GB GDDR5, and Microsoft Windows 10 (64-bit).

The first example presents the use of iterations other than Picard's iteration. In the example we applied the following parameters: $A = [-10, 10]^2$, $K = 50$, a standard convergence test with $\varepsilon = 0.1$ and

$$\begin{cases} f(x,y) = 0.5 \sin 2x |\cos y| + 0.5 \sin x |\cos 2y|, \\ g(x,y) = 0.5 |\cos x| \sin 2y + 0.5 |\cos 0.5x| \sin y. \end{cases} \quad (39)$$

Figure 2 presents patterns obtained with the Picard (a) and Noor (b–d) iterations. The parameters used in the Noor iteration were as follows: (b) $\alpha_n = (-0.5, 0.0)^T$, $\beta_n = (0.5, 0.0)^T$, $\gamma_n = (0.9, 0.0)^T$, (c) $\alpha_n = (0.5, -0.5)^T$, $\beta_n = (1.0, -0.5)^T$, $\gamma_n = (0.1, 0.0)^T$, (d) $\alpha_n = (1.5, 0.0)^T$, $\beta_n = (1.5, 0.0)^T$, $\gamma_n = (-0.5, 0.0)^T$. From the images we see that changing the iteration from Picard to others has a great impact on the shape of the pattern. In this way, we obtain patterns that are similar to the original one (Fig. 2(d)) or that are very different (Figs. 2(b) and (c)). Moreover, we can observe that using vector parameters with the non-zero second coordinate results in some swirls in the pattern (Fig. 2(c)), which were not present in the pattern obtained with the Picard iteration.

The generation times of the images from Fig. 2 are presented in Table 1. From the results we see that the



(a) (b) (c) (d)

Fig. 2. Examples of patterns obtained using the Picard (a) and Noor (b–d) iterations.

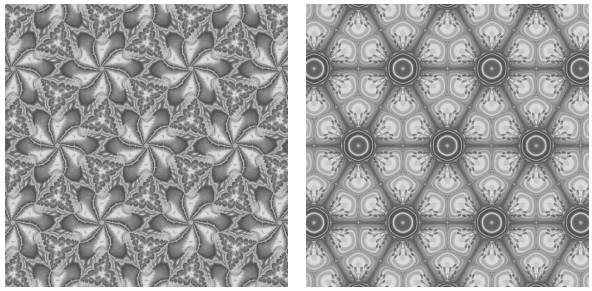
Table 1. Generation times of patterns from Fig. 2.

Figure	CPU time [s]	GPU time [s]	CPU/GPU
2(a)	34.66	0.013	2666.15
2(b)	230.82	0.117	1972.82
2(c)	173.61	0.076	2284.34
2(d)	89.57	0.041	2184.63

generation times for the patterns obtained with the Noor iteration are greater than those for the Picard iteration, even in the case of similar patterns (Figs. 2(a) and (d)). This is due to the fact that the Noor iteration is much more complex, i.e., it requires the function evaluation three times in a single iteration, whereas in the Picard iteration the function's value is calculated only for one point. We can also observe that the values of the parameters in the Noor iteration have a great impact on the generation time. Moreover, we see that the GPU implementation is much faster (about 2000 to 2600 times) than the CPU one. Through the GPU implementation we are able to generate patterns in real time.

The next two examples present the use of the combined iteration process. We start with the example showing the application of a single mapping T used in the combination of two different iterations. The common parameters for generating the patterns in this example were as follows: $A = [-20, 20]^2$, $K = 50$, a standard convergence test with $\varepsilon = 0.1$, the f and g functions defining the mappings $T_1 = T_2$ given by (10).

Figure 3 presents patterns obtained using the



(a) (b)

Fig. 3. Patterns obtained using the S (a) and Mann (b) iterations.

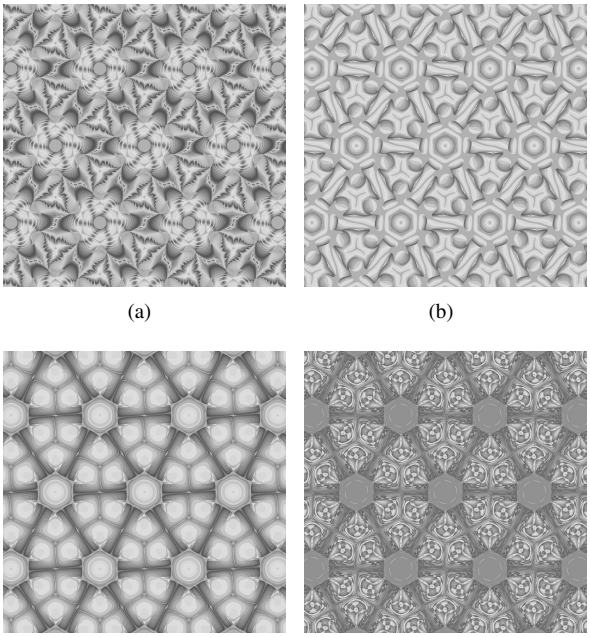
Table 2. Generation times of patterns from Figs. 3 and 4.

Figure	CPU time [s]	GPU time [s]	CPU/GPU
3(a)	217.52	0.172	1264.65
3(b)	121.36	0.080	1517.00
4(a)	198.81	0.147	1352.44
4(b)	71.47	0.049	1458.57
4(c)	101.42	0.064	1584.68
4(d)	230.06	0.211	1090.33

S-iteration with $\alpha_n = (-1.4, 0.0)^T$, $\beta_n = (2.5, 0.5)^T$ (a), and the Mann iteration with $\alpha_n = (3.0, 0.0)^T$ (b). Comparing the images from Figs. 3 and 1(b), we see the next example of using iterations other than the Picard one, because the pattern from Fig. 1(b) was created using the same parameters as the patterns from Fig. 3, but with the Picard iteration. Patterns obtained with the affine combination of the iterations used in Fig. 3 are presented in Fig. 4. The parameters in the affine combination were as follows: (a) $\lambda_1 = 0.8$, $\lambda_2 = 0.2$, (b) $\lambda_1 = 0.6$, $\lambda_2 = 0.4$, (c) $\lambda_1 = 0.2$, $\lambda_2 = 0.8$, (d) $\lambda_1 = -0.2$, $\lambda_2 = 1.2$. From the images, we can observe that the use of the affine combination of two iterations changes the shape of the patterns. Depending on the coefficients of the combination, we obtain patterns that are very similar to one or the other pattern.

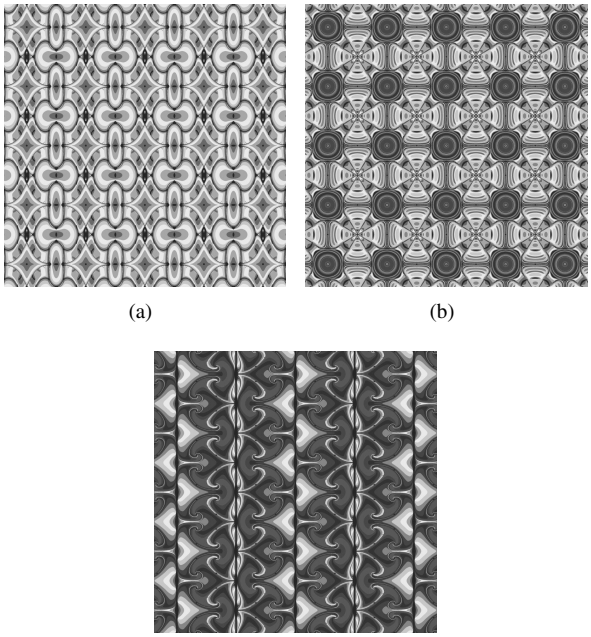
The generation times of the patterns from Figs. 3 and 4 are presented in Table 2. From the obtained results we can observe that the times vary for different values of the parameters of the affine combination, i.e., for the patterns in Figs. 4(b) and (c) the times were lower than the ones obtained for the original patterns in Fig. 3, for the pattern in Fig. 4(d) we observed a longer time, and for the pattern in Fig. 4(a) the time ranges between the generation times of the original patterns. We can also observe that the more complex the pattern, the longer it takes to generate it. Moreover, the GPU implementation is considerably faster than the CPU one, but the speed-up is lower than in the case of the Noor iteration and ranges from about 1000 to 1600.

In the next example we present the use of a



(a) (b) (c) (d)

Fig. 4. Examples of patterns obtained using the affine combination of patterns from Fig. 3.



(a) (b) (c)

Fig. 5. Examples of patterns obtained using the S (a), Mann (b) and Picard (c) iterations.

combination of different mappings and iterations. The common parameters employed in this example were as follows: $A = [-15, 15]^2$, $K = 50$, standard convergence test with $\varepsilon = 0.1$. Figure 5 presents patterns obtained with

(a) the S-iteration with $\alpha_n = (-0.5, 0.0)^T$, $\beta_n = (1.5, 0.0)^T$ and f, g given by (9),

(b) the Mann iteration with $\alpha_n = (1.7, 0.0)^T$ and f, g given by (14),

(c) the Picard iteration and

$$\begin{cases} f(x, y) = \sin 0.5x \cos 2y, \\ g(x, y) = \sin 1.5x \sin y. \end{cases} \tag{40}$$

Likewise, we can observe how the change of the iteration from Picard to others has an impact on the shape of the patterns. Patterns from Figs. 5(a) and (b) were obtained with different iterations (the S and Mann iterations), and the original patterns generated using Picard's iteration are presented in Figs. 1(a) and (d), respectively.

Patterns obtained with the affine combination of the iterations and functions used in Fig. 5 are presented in Fig. 6. The parameters of the affine combination were as follows: (a) $\lambda_1 = 0.19$, $\lambda_2 = 0.8$, $\lambda_3 = 0.01$, (b) $\lambda_1 = 0.7$, $\lambda_2 = 0.2$, $\lambda_3 = 0.1$, (c) $\lambda_1 = \lambda_2 = \lambda_3 = 1/3$, (d) $\lambda_1 = 1.5$, $\lambda_2 = -0.05$, $\lambda_3 = -0.45$. Looking at the images, we can observe that the patterns obtained with the use of the affine combination differ from the original ones, but possess some of their features.

Table 3. Generation times of patterns from Figs. 5 and 6.

Figure	CPU time [s]	GPU time [s]	CPU/GPU
5(a)	82.03	0.043	1907.67
5(b)	119.67	0.067	1786.11
5(c)	179.39	0.092	1949.89
6(a)	92.77	0.050	1855.40
6(b)	104.87	0.055	1906.72
6(c)	162.69	0.090	1807.66
6(d)	224.76	0.159	1413.58

Table 3 presents the generation times obtained for the patterns in Figs. 5 and 6. Similar to the affine combination of two mappings, the times vary for different values of the parameters of the affine combination. Mostly, they range between the times obtained for the patterns that were used in the combination (Figs. 6(a)–(c)), but for the pattern in Fig. 6(d) the time was greater than in the case of the original patterns. Moreover, we see that the speed-up, when using the GPU implementation, varies between about 1400 and 2000, and allows pattern generation in real time.

The last example (Fig. 7) presents the use of different convergence tests. The common parameters employed to generate the patterns were as follows: $A = [-15, 15]^2$, $K = 75$, the Picard iteration and

$$\begin{cases} f(x, y) = \sin x \sin 2y, \\ g(x, y) = \cos x \cos 1.5y. \end{cases} \tag{41}$$

Convergence tests used to obtain the patterns from Fig. 7

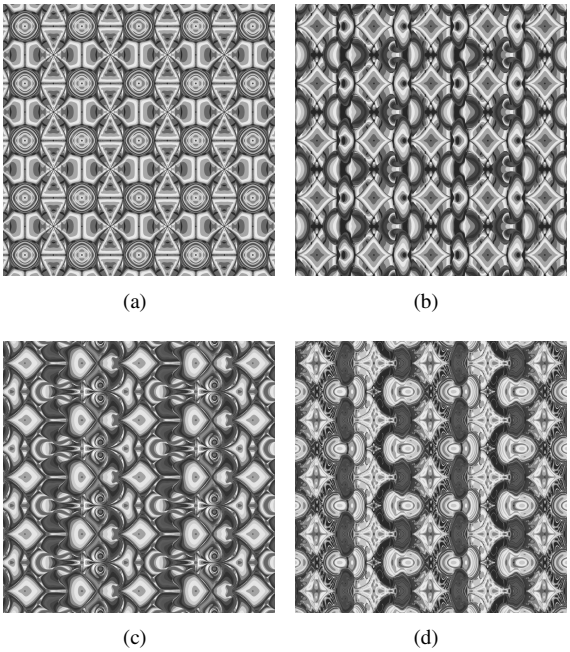


Fig. 6. Examples of patterns obtained using the affine combination of patterns from Fig. 5.

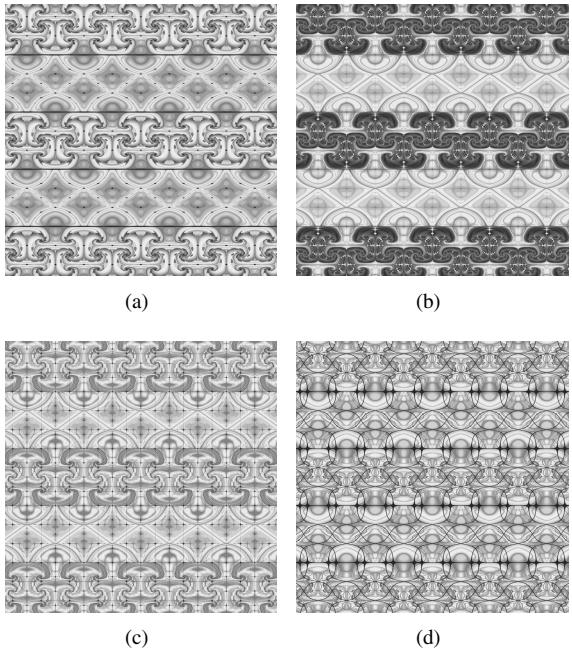


Fig. 7. Examples of patterns obtained using different convergence tests.

were as follows: (a) a standard test with $\varepsilon = 0.1$, (b) a fractional distance test with $q = 0.3$, $\varepsilon = 0.1$, (c) an alternative test with $a = b = 30.0$, $\varepsilon_1 = \varepsilon_2 = 0.1$, (d) difference test with $a = 1.0$, $b = 3.0$, $q = 1.0$, $r = 1.0$, $\varepsilon = 0.1$. From the images in Fig. 7, we see that the change in the convergence test has an impact on the obtained patterns. The use of tests other than the standard one introduces some very interesting details into the obtained patterns, which we had not been able to obtain previously.

The generation times of the patterns from Fig. 7 are presented in Table 4. From the results, we see that the generation times for the non-standard convergence tests are greater compared with the time obtained for the standard test. The difference depends on the complexity and computational cost of the operations used in the test, e.g., in the fractional distance test we have to compute the q-th power and the q-th root, which is computationally expensive. Despite the greater computational cost, however, by using the non-standard tests we were able to obtain new details in the patterns. Similarly to the other examples, we also see in this case that the GPU implementation works in real time. The GPU implementation is about 1800 to 2300 times faster than the CPU one.

Table 4. Generation times of patterns from Fig. 7.

Figure	CPU time [s]	GPU time [s]	CPU/GPU
7(a)	24.11	0.012	2009.16
7(b)	59.20	0.026	2276.92
7(c)	24.03	0.013	1848.46
7(d)	28.38	0.014	2027.14

5. Conclusions

This paper proposed extensions of the method of generating patterns from dynamics using different iteration processes and convergence tests. The application of iterations from fixed-point theory provides more possibilities for obtaining new and very interesting patterns. Moreover, using the proposed generalizations, we are able to obtain patterns which we had not been able to arrive at previously. The proposed methods are also easy to implement using GPU shaders, and this implementation allows the generation of patterns in real time.

In the literature we can find some approaches to obtaining 3D patterns from dynamics (Chung and Chan, 1995; Lu *et al.*, 2012; 2014). As far as future work is concerned, one can attempt the introduction of similar generalizations, like those presented in the paper, to develop algorithms of 3D patterns and introduce new methods. Moreover, it is necessary for the generation of patterns from dynamics to find values of parameters that will produce interesting patterns. This can be very tedious work, which could be made easier by developing automatic methods of finding aesthetic patterns from dynamics.

Acknowledgment

The author would like to thank the anonymous referees for their careful reading of the manuscript as well as their fruitful comments and suggestions that helped improve this paper.

References

Agarwal, R., O'Regan, D. and Sahu, D. (2007). Iterative construction of fixed points of nearly asymptotically nonexpansive mappings, *Journal of Nonlinear and Convex Analysis* **8**(1): 61–79.

Anderson, D. and Wood, Z. (2008). User driven two-dimensional computer-generated ornamentation, *in* G. Bebis *et al.* (Eds.), *Advances in Visual Computing: 4th International Symposium, ISVC 2008. Proceedings, Part I*, Springer, Berlin/Heidelberg, pp. 604–613.

Ashish, Rani, M. and Chugh, R. (2014). Julia sets and Mandelbrot sets in Noor orbit, *Applied Mathematics and Computation* **228**: 615–631.

Chen, Y.-S., Shie, J. and Chen, L.-H. (2012). A NPR system for generating floral patterns based on l-system, *Bulletin of Networking, Computing, Systems, and Software* **1**(1): 38–41.

Chung, K. and Chan, H. (1993). Symmetrical patterns from dynamics, *Computer Graphics Forum* **12**(1): 33–40.

Chung, K. and Chan, H. (1995). Spherical symmetries from dynamics, *Computers & Mathematics with Applications* **29**(7): 67–81.

Chung, K., Chan, H. and Wang, B. (2001). Tessellations in three-dimensional hyperbolic space from dynamics and the quaternions, *Chaos, Solitons & Fractals* **12**(7): 1181–1197.

Ebert, D., Musgrave, F., Peachey, D., Perlin, K. and Worley, S. (2002). *Texturing and Modeling: A Procedural Approach, 3rd Edition*, Morgan Kaufmann, San Francisco, CA.

Gdawiec, K. (2013). Polynomiography and various convergence tests, *in* V. Skala (Ed.), *WSCG 2013 Communication Papers Proceedings*, Vaclav Skala—Union Agency, Plzen, pp. 15–20.

Gdawiec, K. (2017). Inversion fractals and iteration processes in the generation of aesthetic patterns, *Computer Graphics Forum* **36**(1): 35–45.

Gdawiec, K. and Kotarski, W. (2017). Polynomiography for the polynomial infinity norm via Kalantari's formula and nonstandard iterations, *Applied Mathematics and Computation* **307**: 17–30.

Gdawiec, K., Kotarski, W. and Lisowska, A. (2015). Polynomiography based on the non-standard Newton-like root finding methods, *Abstract and Applied Analysis* **2015**, Article ID: 797594.

Greenfield, G. (2016). Turing-like patterns from cellular automata, *in* E. Torrence *et al.* (Eds.), *Proceedings of Bridges 2016: Mathematics, Music, Art, Architecture, Education, Culture*, Tessellations Publishing, Phoenix, AZ, pp. 151–158.

Horne, C. (2000). *Geometric Symmetry in Patterns and Tilings*, CRC Press, Boca Raton, FL.

Jia, C. and Ming-Xi, T. (2013). Integrating shape grammars into a generative system for Zhuang ethnic embroidery design exploration, *Computer-Aided Design* **45**(3): 591–604.

Kang, S., Alsulami, H., Rafiq, A. and Shahid, A. (2015a). S-iteration scheme and polynomiography, *Journal of Nonlinear Science and Applications* **8**(5): 617–627.

Kang, S., Rafiq, A., Latif, A., Shahid, A. and Kwun, Y. (2015b). Tricorns and multicorns of S-iteration scheme, *Journal of Function Spaces* **2015**, Article ID: 417167.

Klempien-Hinrichs, R. and von Totth, C. (2010). Generation of Celtic key patterns with tree-based collage grammars, *Electronic Communications of the EASST* **26**: 205–221.

Lalitha, D. and Rangarajan, K. (2012). Petri nets generating Kolam patterns, *Indian Journal of Computer Science and Engineering* **3**(1): 68–74.

Lu, J., Ye, Z. and Zou, Y. (2007). Automatic generation of colorful patterns with wallpaper symmetries from dynamics, *The Visual Computer* **23**(6): 445–449.

Lu, J., Zou, Y. and Li, W. (2010). Colorful patterns with discrete planar symmetries from dynamical systems, *Fractals* **18**(1): 35–43.

Lu, J., Zou, Y., Liu, Z. and Li, W. (2012). Colorful symmetric images in three-dimensional space from dynamical systems, *Fractals* **20**(1): 53–60.

Lu, J., Zou, Y., Yang, C. and Wang, L. (2014). Orbit trap rendering methods for generating colorful symmetric images in three-dimensional space, *Nonlinear Dynamics* **77**(4): 1643–1651.

Mann, W. (1953). Mean value methods in iteration, *Proceedings of the American Mathematical Society* **4**(3): 506–510.

Noor, M. (2000). New approximation schemes for general variational inequalities, *Journal of Mathematical Analysis and Applications* **251**(1): 217–229.

Ouyang, P., Zhao, W. and Huang, X. (2015). Beautiful math. Part 5: Colorful Archimedean tilings from dynamical systems, *IEEE Computer Graphics and Applications* **35**(6): 90–96.

Pickover, C. (2001). *Computers, Pattern, Chaos, and Beauty: Graphics from an Unseen World*, Dover Publications, Mineola, NY.

Qi, W. and Li, X. (2009). Example-based floral pattern generation, *Proceedings of the 5th International Conference on Image and Graphics, Xi'an, Shanxi, China*, pp. 553–558.

Sayed, Z., Ugail, H., Palmer, I., Purdy, J. and Reeve, C. (2016). Auto-parameterized shape grammar for constructing Islamic geometric motif-based structures, *in* M. Gavrilova *et al.* (Eds.), *Transactions on Computational Science XXVIII: Special Issue on Cyberworlds and Cybersecurity*, Springer, Berlin/Heidelberg, pp. 146–162.

Setti, R. (2015). Generative dreams from deep belief networks, *in* C. Soddu and E. Colabella (Eds.), *Generative Art 2015: Proceeding of the XVIII Generative Art Conference*, Domus Argenia Publisher, Milan, pp. 260–273.

von Gagern, M. and Richter-Gebert, J. (2009). Hyperbolization of Euclidean ornaments, *Electronic Journal of Combinatorics* **16**(2): R12.

Wei, L.-Y., Lefebvre, S., Kwatra, V. and Turk, G. (2009). State of the art in example-based texture synthesis, *State of the Art Report: EG-STAR*, Eurographics Association, Munich.

Yeh, Y.-T., Breeden, K., Yang, L., Fisher, M. and Hanrahan, P. (2013). Synthesis of tiled patterns using factor graphs, *ACM Transactions on Graphics* **32**(1), Article no. 3.

Krzysztof Gdawiec received his MSc degree in mathematics from the University of Silesia (Poland) in 2005, and his PhD degree in computer science from the same university in 2010. He is currently employed as an assistant professor at the Institute of Computer Science of the University of Silesia, and is an author of several journal and conference publications. His main research interests include applications of fractal geometry, pattern recognition and computer graphics. He is a member of the Polish Mathematical Society and SIGGRAPH.

Received: 20 April 2017
Revised: 30 July 2017
Accepted: 1 September 2017